Uncovering discourse structure in program summaries

Uncovering discourse structure in program summaries

Jon Oberlander and Judith Good

Division of Informatics, University of Edinburgh

Abstract

We have been analysing systematic differences between the summaries of computer programs produced by two groups of students. Each group has been exposed to one of two visual programming languages with different underlying paradigms: control flow, and data flow. The study builds on Pennington's approach to information types in program summaries, developing a slightly more ramified set of information types. Using this framework, we gathered, annotated and analysed a small corpus of 80 summaries. A simple, covering set of discourse relations emerges from the data, by co-classifying subsets of the information types, and charting their patterns in the summaries. The resulting characterisation of the groups' discourses can be used to show that control flow subjects exhibit relatively unvaried structure in their summaries, compared with data flow subjects.

Uncovering discourse structure in program summaries

1 Introduction

People sometimes have to describe quite complex artefacts to other people. Amongst these artefacts are computer programs, which can possess strange structure and exhibit bewildering behaviours. What people say about a particular program can be influenced by the way they originally saw it, as well as the underlying programming paradigm with which they are familar.

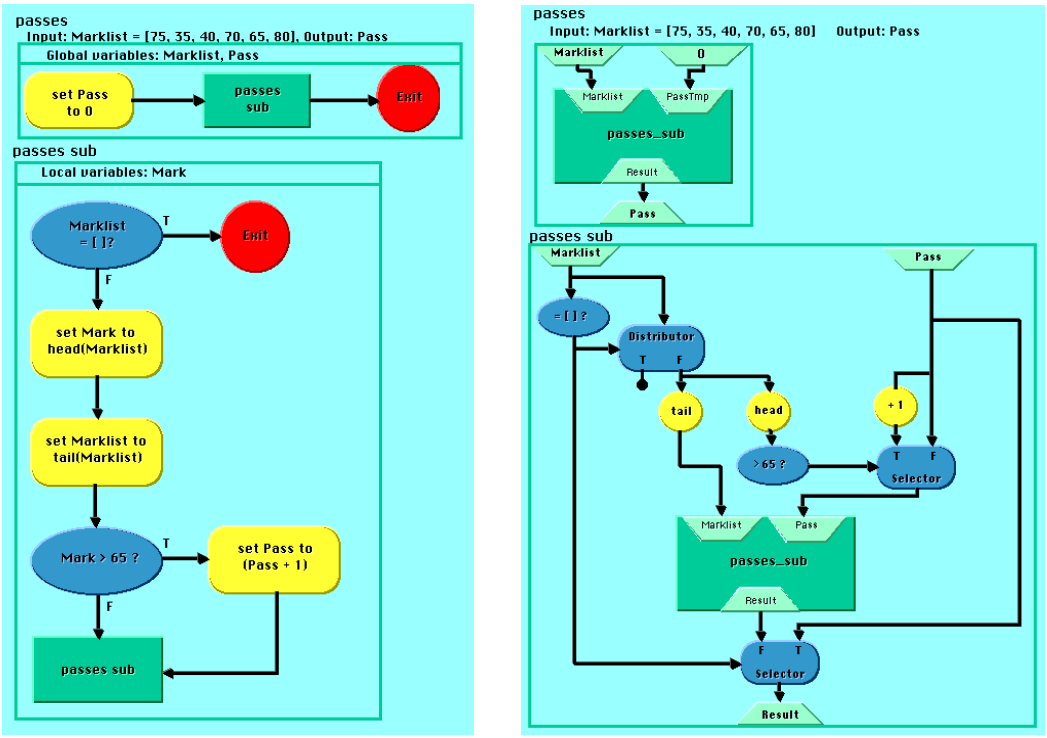

Figure 1: The `passes` program. Left: control flow version. Right: data flow version.

For instance, the diagrams in Figure 1 portray the same program seen in control flow or data flow terms. Two students summarised it thus (spelling errors corrected):

(1) it first checks to see if the mark list is empty if it is then the program exits. if that check is false it then sets mark to the first number in the list, and sets the rest of the list to some variable, it then checks to see if mark is greater than 65 if it is it then adds 1 to pass and exit if it is not it recurses

(2) This is processing a list of marks and finding which are greater than 65 and adding 1 to the counter to give a result of how many passed

These two descriptions were written by two different students, but they were both describing the `passes` program. By eliciting program summaries like these, we have been

analysing systematic differences between the summaries produced by groups of students, where the groups have been exposed to two visual programming languages with different underlying paradigms. One language reflects the control flow paradigm; the other reflects the data flow paradigm.

This work leads us to address two of the key questions set for the workshop:

- Reader properties and microstructure: How does the reader's background knowledge interact with the structure and wording of the document?
- What kind of coherence relations are there? In particular, how can they be grouped?

We focus here on properties of the writers, rather than the readers, but with this proviso, we wish to argue that our analysis of the discourse structures of program summaries offers insights relevant to these initial questions.

In particular, we will argue that exposure to a particular visual programming paradigm causes people to communicate their knowledge of the program in characteristic ways—exemplifed in texts (1) and (2). To show this, we will exploit a simple statistical approach to discourse structure which makes an appropriate, covering set of discourse relations emerge directly from: (i) the data itself, combined with (ii) a defined set of statement categories, and (iii) a simple criterion for organising those categories.

To this end, the rest of the paper is structured as follows. The next section introduces some background on Pennington's approach to information types in program summaries. Section 3 describes the experimental regime under which we gathered a small corpus of program summaries, and Section 4 outlines the specific annotation scheme for information types used to analyse these summaries. Section 5 furnishes general results, before detailing those arising from the information type analysis; a small, covering set of discourse relations emerges from the data, by co-classifying subsets of the information types, and charting their bigram patterns in the summaries.

2 Background

A definition of 'information types' in programming appears to date from a paper by Pennington in which she described them as 'different kinds of information explicit 'in the text' that must be detected in order to fully understand the program' [6, p. 299]. The following five information types were identified:

Function: information about the overall goal of the program, essentially, "What is the purpose of the program? What does the program *do*?"

Control flow: information about the temporal sequence of events occurring in the program, *e.g.* "What happens just before/after X occurs?

Data flow: concerned with the data transformations, data dependencies and data structure information, *e.g.* "Does variable X contribute to the final value of Y?"

Operations: information about specific, low-level actions which occur in the code, such as "Does a variable become instantiated with a particular value?"

State: time-slice descriptions of the state of objects and events in the program, *e.g.* "When the program is in state X, is event Y taking place?"

Pennington used information types to investigate the nature of programmers' mental representations. She based her work on the theory of text comprehension put forward by Kintsch and van Dijk [3, 7], whereby text comprehension results in the production of two distinct but interrelated representations of the text: the *textbase* and the *situation model*. Similarly, Pennington hypothesised that when 'reading' a computer program, the textbase is first built, based on a procedural reading of the program, and couched in terms of the programming language. The situation model (or 'program model' as Pennington calls it) is built from this textbase, and highlights functional relationships between domain objects.

It follows that program comprehension occurs from the bottom up: programmers first divide the program into small control segments, making inferences about each segment's procedural role. Function and data flow run through segments, and are thought to be uncovered through a process of integration later in the comprehension process.

We chose to investigate this hypothesis using two programming paradigms, control flow and data flow, which are based on markedly different conceptualisations of the programming process. In the former, the program is seen as a series of instructions which are executed in a predetermined order, using data as and when necessary. In the latter, the focus is on data as a dynamic entity, flowing through a program and triggering actions by its presence.

This opposition between control and data, and between processes and objects, runs through human language more generally, and is undoubtedly important when we talk about complex artefacts. Here, we hypothesised that the differences inherent in the programming paradigms could lead to characteristically different models of comprehension observable via the information types used in program summaries. We will return to information types once we have described the experimental regime.

3 Experiment

The study we carried out[1] examined the effect of the visual programming language paradigm (data flow or control flow) on program comprehension, as measured by multiple choice questions (functional, data flow, control flow, operations and state), and a short program summary. The programs used were short, recursive list processing programs.

Twenty two participants took part in the experiment (although data from two participants had to be discarded as they experienced difficulties during the experiment). All were starting the second year of a computer studies degree, and had been taught C^{++} and COBOL. The experiment included a number of pre-tests which explored graphical skill and prior programming knowledge: as the results are not reported in this paper, they will not be described.

Controlling for expertise, subjects were divided randomly into two groups: control flow and data flow. The experimental setup was implemented in Macromedia Director. The first

[1] Described in full in [2].

screen explained the overall structure of the experimental session. The following screens provided an introduction to the visual programming language, presenting the nodes used in the language with a textual description of their function. Note that these introductory screens necessarily differed between the two groups. Participants also saw a sample program, with an explanation of how each node contributed to producing the program output. A practice session allowed participants to answer questions similar to those they would encounter in the experiment.

In the experiment itself, participants were asked to study a short program. The next screen required subjects to type a free-form summary of the program into a text box (the program was not visible on this screen). The subsequent five screens each showed a multiple choice comprehension question (presented in random order), corresponding to one of the five information types, along with the program. Each group saw the same programs, but their form differed depending upon the group to which the subject had been assigned: control flow or data flow. Participants worked through a total of four programs. Therefore with 20 subjects producing four summaries each, we gathered 80 program summaries in total.

4 Analysis scheme

As just described, after studying a program, subjects were asked to write a short summary. The instructions were non-directive so as to allow them to describe the program in their own words. The aim was to determine whether there might be differences between groups in terms of how they described the programs. Pennington [6] first analysed summaries of this type. However, the description of her scheme was lacking in the detail necessary to replicate it. Therefore, a new scheme was developed by refining the category definitions, providing fuller definitions and examples, and extending Pennington's three categories (function, control and data) to ten, which are described below with examples.

function the overall aim of the program, described succinctly.
– The program is selecting all players over a certain height and allowing then to join the team.

actions events described at a lower level than function, but at a higher level than operations (described below). An action may involve a small group of nodes rather than one node only, operate over several inputs, or describe actions in non-specific ways.
– This sub-program checks each individual element of this list ...

operations small-scale events, usually corresponding to one node/line of code.
– ... then the program sets the height to head(height).

state-high a high-level definition of state which differs from state-low in terms of granularity: state-high describes an event at a more abstract level. The relationship between the two is akin to the relationship between actions and operations.
– Once all the elements have been processed ...

state-low a lower-level version of state-high. State-low often relates to a test condition being met, or not met, and upon which an operation depends.
– If the head is greater than 65 ...

Table 1: Low level description of the `passes` program, with statement types.

Statement	Code
it first checks to see if the mark list is empty	operation
if it is	state – lo
then the program exits.	control
if that check is false	state – lo
it then sets mark to the first number in the list,	operation
and sets the rest of the list to some variable,	operation
it then checks to see if mark is greater than 65	operation
if it is	state – lo
it then adds 1 to pass	operation
and exit	control
if it is not	state – lo
it recurses	control

data inputs and outputs to programs, data flow through programs, descriptions of data objects and data states.
– . . . it then passes a list of heights to a sub-program.

control program control structures and/or sequencing, *e.g.* recursion, calls to subprograms.
– It exits the program and goes back to the main program.

elaborate further information about an already described process/event/data object (including concrete examples).
– [If the current mark is above a certain pass level] (65 in this case) . . .

meta statements about the participant's own reasoning.
– Dhoo! forgot where that route went!!!

unclear statements which are ambiguous or uninterpretable.
– [If the height is greater than 180, 1 is added to the counter] and the height is recorded.
It is not clear here whether 'recorded' means 'printed', 'added to a list', 'assigned to a variable', or something else.

The application of the scheme is illustrated using the program descriptions which we started with: (1) and (2) are presented, with annotations, in Tables 1 and 2.

5 Results

5.1 Summary of overall results

Analysis of subjects' behaviour and responses suggest that the data flow language presented more difficulties for novices. In particular, data flow subjects spent significantly

Table 2: High level description of the `passes` program, with statement types.

Statement	Code
This is processing a list of marks	action
and finding which are greater than 65	action
and adding 1 to the counter	action
to give a result of how many passed	data

Table 3: Mean Proportion of Information Types Statements per Group

Category	Ctrl Flow	Data Flow
	Mean %	Mean %
function	11.62	20.93
action	7.10	9.10
operation	30.22	15.67
state-high	6.22	8.23
state-low	12.93	10.04
data	13.10	24.68
control	14.10	5.33
elaborate	.49	3.61
meta	.15	1.05
unclear	4.07	1.36

longer learning the micro-language, inspecting the experimental programs and answering the multiple choice questions than did the control flow subjects. Overall, subjects who studied control flow programs tended to perform better on questions requiring information about the details of a program, or the order in which events occur—in short, how the program works. Those who saw data flow programs tended to score higher on questions requiring higher level, more abstract knowledge about what the program does.

5.2 Information types in program summaries

The mean length of the summaries was 70.91 words for the control flow group, and 48.85 words for the data flow group. However, this difference was not significant, owing to a high degree of variance. Table 3 shows the mean proportion of information types category statements for the control and data flow groups. Summaries from the data flow group contain higher proportions of function, action, state-high, and data flow information types than do the control flow group. The control flow group's summaries contain higher proportions of operation, state-low, and control flow statements.

A mixed design ANOVA for repeated measures with groups as a 2 level between-subjects factor and statement type as a 10 level, repeated measures, factor, showed a significant effect for statement type ($F=7.19$, $df(9,162)$, $p < .001$), and a significant group by statement interaction ($F=1.94$, $df(9,162)$, $p < .05$). Four post-hoc pairwise comparisons (function,

operations, data, control), made using the Bonferonni adjustment, showed a significant effect for the control flow comparison (t= 2.62, p < .01).

One issue of interest is the *level of abstraction* of the information type. It is possible to group the information types into three main levels of abstraction: the highest level (here designated level 1) includes function, elaborate and meta statements; the intermediate level (level 2) includes action, state-high and data statements; the lowest genuine level (level 3) includes operation, state-low and control statements; finally, level 4 is reserved for unclear statements. With this grouping of information types in hand, we can now examine the kinds of sequential patterns in which they occur, through corpus analysis.

5.3 Patterns of information types

Following Oberlander et al. [4], we apply to discourse a set of techniques more usually applied to words or parts of speech tags. Many statistical tests of natural language assume a normal distribution of units within the corpus of language. For various reasons, this assumption is too strong. In a relatively small corpus like ours, containing 80 program summaries, there is no guarantee that there will be 'standard' occurrences of all units. Of course, one could focus only on the most frequent patterns within the corpus. But this means that 'anomalies' are ignored, and these may be the most important elements in determining individual or group style. The 'Log–Likelihood Ratio' described in [1] addresses these problems;[2] it is designed to 'highlight particular A's and B's that are highly associated in text' (p.71), and it is claimed that its results accord well with intuitions regarding the naturalness (or otherwise) of the relevant bigrams. Ranking the bigrams according to this test provides a good *profile* of a group's use of information types in their program summaries.

Tables 5.3 and 5 show the significantly associated bigrams for the control flow and data flow groups, respectively. These profiles reveal that the program summaries produced by the two groups differ at a deeper level than mere word count, or proportion of statements of a given information type. The *patterns* of information types are quite distinctive.

First, the control flow and data flow groups had 19 and 11 significantly associated types of bigram respectively. Of these, only 4 were common to both groups. The operation–data and control–state-high bigrams were the top two bigrams for data flow subjects, but appear as the last two significant bigrams for the control flow group. The state-high–operation and operation–action bigrams were also 'shared'. Thus, it seems reasonable to conclude that the groups are exhibiting differing sequential patterns of use of the information types.

Secondly, when we consider the levels of abstraction involved, a very clear distinction emerges. Inspection of Tables 5.3 and 5 suggests that control flow subjects produce bigrams which rarely change the level of abstraction in the summary: hence the relatively large number of zeroes in the final column of the table. By contrast, data flow subjects seem to make more use of level-changing bigrams, moving up or down one level of abstraction at a time: hence the 1s and –1s in their final column. This subjective impression is confirmed when we compute the overall patterns of level change in the two groups' summaries.

[2]Some have argued that Fisher's exact test is preferable to the log-likelihood ratio [5]; we therefore repeated our analysis using Fisher's exact test; the results differ in only minor details.

Table 4: Bigram profile for control flow subjects' program summaries. The first column indicates the 'log-likelihood ratio', a measure of the significance of the distribution of a particular bigram in the corpus. $k(AB)$ is a count of the number of times the bigram AB occurs, $k(A \sim B)$ is a count of the number of times A is followed by an information type other than B, and so on. $l(A)$ and $l(B)$ give the respective unigrams' levels of abstraction; $l(A - B)$ gives the resulting bigram's change in level of abstraction. Only bigrams which are significantly associated ($p < .05$) are shown.

$-2\log\lambda$	k(AB)	k(A∼B)	k(∼AB)	k(∼A∼B)	A	B	l(A)	l(B)	l(A-B)
23.04	9	16	12	304	data	data	2	2	0
22.54	42	24	87	188	state-lo	operation	3	3	0
11.96	39	98	27	177	operation	state-lo	3	3	0
11.36	4	62	62	213	state-lo	state-lo	3	3	0
10.16	21	39	45	236	control	state-lo	3	3	0
9.55	2	7	2	330	function	function	1	1	0
7.88	2	135	16	188	operation	action	3	2	1
7.49	5	20	13	303	data	action	2	2	0
7.35	2	5	6	328	unclear	unclear	4	4	0
7.12	3	134	18	186	operation	state-high	3	2	1
6.15	4	21	125	191	data	operation	2	3	–1
6.09	2	15	127	197	state-high	operation	2	3	–1
5.89	4	13	17	307	action	state-high	2	2	0
5.89	4	13	17	307	state-high	state-high	2	2	0
5.80	37	100	33	171	operation	control	3	3	0
5.65	6	54	64	217	control	control	3	3	0
5.51	1	24	65	251	data	state-lo	2	3	–1
5.33	8	52	13	268	control	state-high	3	2	1
4.57	4	133	17	187	operation	data	3	2	1

Table 5: Bigram profile for data flow subjects' program summaries.

$-2\log\lambda$	k(AB)	k(A∼B)	k(∼AB)	k(∼A∼B)	A	B	l(A)	l(B)	l(A–B)
13.10	4	45	61	132	operation	data	3	2	1
12.78	8	10	22	202	control	state-high	3	2	1
6.13	16	20	49	157	state-lo	data	3	2	1
3.40	2	25	45	170	state-high	operation	2	3	–1
3.04	2	58	18	164	data	control	2	3	–1
5.03	1	0	19	222	meta	action	1	2	–1
3.01	1	8	2	231	elaborate	meta	1	1	0
1.87	13	36	34	159	operation	operation	3	3	0
1.78	2	14	9	217	function	elaborate	1	1	0
1.71	4	56	5	177	data	function	2	1	1
1.65	2	47	18	175	operation	action	3	2	1

Table 6: Mean proportion of types of change of level of abstraction, per group. Note that for control flow, there were 341 bigram tokens in total, of which 199 were significantly associated. For data flow, there were 242 in total, of which 87 were significantly associated.

Switch	**Control Flow**		**Data Flow**	
l(A-B)	**All %**	**Sig. %**	**All %**	**Sig. %**
–3.00	.00	.00	.41	.00
–2.00	2.05	.00	2.07	.00
–1.00	9.38	3.52	25.21	24.14
0.00	78.59	87.94	46.69	42.53
1.00	8.80	8.54	21.90	32.18
2.00	1.17	.00	3.31	.00
3.00	.00	.00	.41	1.15

Table 5.3 indicates the proportions of level changes for the two groups, and provides the figures both for the complete set of bigram instances used in the two groups, and also for the set of instances of significantly associated bigrams.

It emerges that control flow subjects tend not to change level between statements: almost 80% of all their bigrams, and almost 90% of their significant bigrams, represent cases where two consecutive statements may be of differing information type, but are at the same level of abstraction. By contrast, for data flow subjects, only 47% of their bigrams, and just 42% of their significant bigrams, are of this kind. For both groups, the remainder of their bigrams are evenly split between movement up or down one level of abstraction. For instance, in the data flow group, the three top ranked bigrams (operation–data, control–state-high and state-lo–data) move up the abstraction hierarchy, and the next three move down (state-high–operation, data–control and meta–action). Thus, the bigram statistics indicate that control flow subjects change level between statements around one time in five, whereas data flow subjects are at least as likely to change level as to stay within level.

It follows that a complete, covering model of discourse structure in this domain need only posit three types of discourse relation, holding between adjacent pairs of statements. To cover staying within a level of abstraction, raising level and reducing it, these might respectively be termed Continue, Generalise, and Specialise. Their patterns of occurrence, reflecting the bigrams uncovered in the corpus analysis, cleanly distinguish the discourses generated by our two groups of subjects.

6 Conclusion

We can now return to the two workshop questions we raised at the outset.

First, we found that exposure to a particular visual programming paradigm causes people to communicate their knowledge of the program in different ways. So, the real world

visual representation, and the mental representation of it, caused the writers in our two groups to arrive at differing communication strategies. Thus, their background knowledge directly influences the structure and wording of their documents.

Control flow subjects gave quite lengthy, low-level, sequential accounts of the program's execution, while data flow subjects gave shorter, more succinct summaries which were couched at a higher level of abstraction (for instance, describing the program's function). Providing a low-level account of a program necessarily requires more statements: for example, each node in a visual programming language can be described by one or more low-level statements. On the other hand, a high-level account may be able to encompass the entire program in a few statements. Given this, the corpus analysis suggests that once the control flow people choose to provide a low-level statement in their summary, they are more or less constrained to continue at that level in order to provide a full account of the program. By contrast, the data flow people, in providing higher level statements, retain the liberty to move between levels when they wish.

Secondly, we found three discourse relations would suffice for our domain: Continue, Generalise and Specialise. While these might not be so informative for other domains, we would urge that the approach to discourse structure adopted here may have wider appeal. It has the virtue of making an appropriate, covering set of discourse relations emerge directly from: (i) the data itself, combined with (ii) a defined set of statement categories, and (iii) a simple criterion for organising those categories, here in terms of abstractness.

References

[1] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19:61–74, 1993.

[2] J. Good. *Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1999.

[3] W. Kintsch and T. A. van Dijk. Toward a model of text comprehension and production. *Psychological Review*, 85:363–394, 1978.

[4] J. Oberlander, P. Monaghan, R. Cox, K. Stenning, and R. Tobin. Unnatural language processing: an empirical study of multimodal proof styles. *Journal of Logic, Language and Information*, 8:363–384, 1999.

[5] T. Pedersen. Fishing for exactness. In *Proceedings of the South Central SAS User's Group (SCSUG-96) Conference*, pages 188–200, Austin, TX, October 1996.

[6] N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19:295–341, 1987.

[7] T. A. van Dijk and W. Kintsch. *Strategies of Discourse Comprehension.* Academic Press, New York, 1983.